Name: Shazim Moin Khot
Class: SE DIV A
AIML
Roll No: 25AI57

EXP 3.A

AIM:
Implement a Queue using Arrays.

THEORY:-
A Queue is a linear data structure that follows the **FIFO (First In, First Out)** principle. The element inserted first is removed first. The two basic operations performed on a queue are **Enqueue** (insertion) and **Dequeue** (deletion).
In an array implementation of a queue, two variables named **front** and **rear** are used to keep track of the first and last elements. Initially, both are set to **-1**. During insertion, the rear pointer is incremented, and during deletion, the front pointer is incremented.

Queue operations include:-
- Enqueue – Inserts an element at the rear.
- Dequeue – Removes an element from the front.
- Peek – Displays the front element.
- Display – Shows all elements of the queue.

This implementation is simple and efficient but may lead to **memory wastage** because deleted positions cannot be reused once the rear reaches the end of the array.

Code:-
```
#include <stdio.h>
#define MAX 5

int queue[MAX];
int front = -1, rear = -1;
```

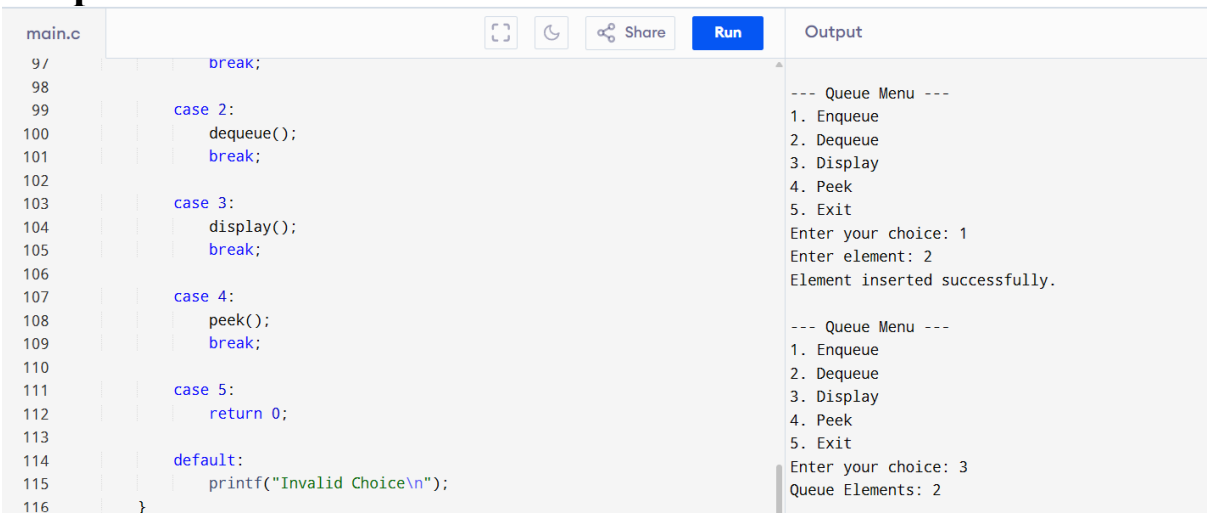
```
void enqueue()
{
   int item;

   if (rear == MAX - 1)
   {
      printf("Queue Overflow\n");
      return;
   }

   printf("Enter element: ");
   scanf("%d", &item);

   if (front == -1)
      front = 0;

   rear++;
   queue[rear] = item;

   printf("Element inserted successfully.\n");
}

void dequeue()
{
   if (front == -1 || front > rear)
   {
      printf("Queue Underflow\n");
      return;
   }

   printf("Deleted element: %d\n", queue[front]);
   front++;

   if (front > rear)
   {
      front = rear = -1;
```

```
    }
}

void display()
{
    int i;

    if (front == -1)
    {
        printf("Queue is Empty\n");
        return;
    }

    printf("Queue Elements: ");

    for (i = front; i <= rear; i++)
    {
        printf("%d ", queue[i]);
    }

    printf("\n");
}

void peek()
{
    if (front == -1)
    {
        printf("Queue is Empty\n");
        return;
    }

    printf("Front Element: %d\n", queue[front]);
}

int main()
{
    int choice;
```

```
while (1)
{
    printf("\n--- Queue Menu ---\n");
    printf("1. Enqueue\n");
    printf("2. Dequeue\n");
    printf("3. Display\n");
    printf("4. Peek\n");
    printf("5. Exit\n");

    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice)
    {
        case 1:
            enqueue();
            break;

        case 2:
            dequeue();
            break;

        case 3:
            display();
            break;

        case 4:
            peek();
            break;

        case 5:
            return 0;

        default:
            printf("Invalid Choice\n");
    }
```

```
    }
}
```

Output:-


main.c
Share
Run
Output
break;
case 2:
dequeue();
break;
case 3:
display();
break;
case 4:
peek();
break;
case 5:
return 0;
default:
printf("Invalid Choice\n");
}
--- Queue Menu ---
1. Enqueue
2. Dequeue
3. Display
4. Peek
5. Exit
Enter your choice: 1
Enter element: 2
Element inserted successfully.
--- Queue Menu ---
1. Enqueue
2. Dequeue
3. Display
4. Peek
5. Exit
Enter your choice: 3
Queue Elements: 2

main.c
Share
Run
Output
break;
case 2:
dequeue();
break;
case 3:
display();
break;
case 4:
peek();
break;
case 5:
return 0;
default:
printf("Invalid Choice\n");
}
Queue Elements: 2
--- Queue Menu ---
1. Enqueue
2. Dequeue
3. Display
4. Peek
5. Exit
Enter your choice: 2
Deleted element: 2
--- Queue Menu ---
1. Enqueue
2. Dequeue
3. Display
4. Peek
5. Exit
Enter your choice: 3
Queue is Empty

Programiz
C Online Compiler
main.c
Share
Run
Output
#include <stdio.h>
#define MAX 5
int queue[MAX];
int front = -1, rear = -1;
void enqueue()
{
int item;
if (rear == MAX - 1)
{
printf("Queue Overflow\n");
return;
3. Display
4. Peek
5. Exit
Enter your choice: 3
Queue is Empty
--- Queue Menu ---
1. Enqueue
2. Dequeue
3. Display
4. Peek
5. Exit
Enter your choice: 2
Queue Underflow

Conclusion:-
The program successfully implements a Queue using an array. It demonstrates the FIFO principle by inserting elements at the rear and deleting them from the front. The experiment helps in understanding queue operations such as enqueue, dequeue, display, and peek. It also highlights the limitation of a linear queue, where unused spaces cannot be reused after deletion. The implementation is simple, efficient, and easy to understand.

EXP 3.B

AIM:-
Implement a Circular Queue using Arrays.

THEORY:-
A Circular Queue is an improved version of a linear queue in which the last position is connected to the first position, forming a circle. It also follows the **FIFO (First In, First Out)** principle.
In a circular queue, when the rear reaches the last index of the array, it wraps around to the beginning if there is free space available. This eliminates the memory wastage problem found in a linear queue.
The circular queue uses two pointers:

- **Front** – Points to the first element.
- **Rear** – Points to the last element.

The queue is considered:-

- **Full** when (rear + 1) % SIZE == front
- **Empty** when front == -1

Operations performed on a circular queue include enqueue, dequeue, display, and peek. Circular queues are commonly used in CPU scheduling, buffering, and real-time systems because they utilize memory efficiently.

Code:-

```
#include <stdio.h>
#define MAX 5

int queue[MAX];
int front = -1, rear = -1;

void enqueue()
{
   int item;

   if ((rear + 1) % MAX == front)
   {
      printf("Queue Overflow\n");
      return;
   }

   printf("Enter element: ");
   scanf("%d", &item);

   if (front == -1)
   {
      front = rear = 0;
   }
   else
   {
      rear = (rear + 1) % MAX;
   }

   queue[rear] = item;

   printf("Element inserted successfully.\n");
}

void dequeue()
{
   if (front == -1)
```

```
    {
        printf("Queue Underflow\n");
        return;
    }

    printf("Deleted element: %d\n", queue[front]);

    if (front == rear)
    {
        front = rear = -1;
    }
    else
    {
        front = (front + 1) % MAX;
    }
}

void display()
{
    int i;

    if (front == -1)
    {
        printf("Queue is Empty\n");
        return;
    }

    printf("Queue Elements: ");

    i = front;

    while (1)
    {
        printf("%d ", queue[i]);

        if (i == rear)
            break;
```

```
        i = (i + 1) % MAX;
    }

    printf("\n");
}

void peek()
{
    if (front == -1)
    {
        printf("Queue is Empty\n");
        return;
    }

    printf("Front Element: %d\n", queue[front]);
}

int main()
{
    int choice;

    while (1)
    {
        printf("\n--- Circular Queue Menu ---\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Display\n");
        printf("4. Peek\n");
        printf("5. Exit\n");

        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
            case 1:
```

```
            enqueue();
            break;

        case 2:
            dequeue();
            break;

        case 3:
            display();
            break;

        case 4:
            peek();
            break;

        case 5:
            return 0;

        default:
            printf("Invalid Choice\n");
        }
    }
}
```

Output:-

Conclusion:-

The program successfully implements a Circular Queue using an array. It efficiently utilizes memory by reusing vacant positions created after deletion. The experiment demonstrates queue operations such as enqueue, dequeue, display, and peek while overcoming the limitation of a linear queue. Circular queues provide better memory management and are widely used in operating systems and real-time applications.